Linear Algebra Assignment 2 Report

110062219 翁君牧

November 16, 2022

Contents

1 Different Selections of Equations **2**

2 Using *Linear Least Square* **2**

3 Take a Photo of a Rectangular Object **3**
3.1 Compute the Projection Matrix 3
3.2 Draw the 3D Model 4

4 Algorithm Related to Single View Geometry **5**
4.1 Derivations 5
4.2 Processes 6

5 Automatically Select the Corresponding Points **6**
5.1 Procedures 6
5.2 Outcomes 7

List of Figures

1 The shoe box 3
2 The 3D model of the shoe box 4
3 The screenshot of a block in Minecraft 7
4 The detected corners, or vertices of the block are dotted . . . 8
5 The 3D model of the block 8

1 Different Selections of Equations

It's obvious that there are $\binom{14}{11} = 364$ combinations. As a consequence, it's not hard to enumerate all selections and measure, compare their performances. I allocated a full single node on **Taiwania 3** to run the Jupyter server.

Each one was iterated 10000 times in loop and repeat 5 times. We should be cautious that some combinations are *singular* which means that it couldn't lead to a solution.

The minimum was $9.08\mu s$ whereas the maximum being $9.84\mu s$. Note that even though the amount of iterations had been increased again and again, the result still varied and diverged. That is to say, we failed to obtain an optimal combination that is ensured to yield the best result all times. Still, we could see that it seems that if we select rows 8, 13 then the result have a tendency to be a bit better.

2 Using *Linear Least Square*

We're asked to try to call `numpy.linalg.lstsq()` to solve the overdetermined system. I also iterated it for 10000 times in loop and repeat 5 times so as to do comparison.

We could find that the CPU time consumed by *linear least square*, $38.9\mu s$, was about 4 times the slowest one in different selections of equations. So why's the reason? I sought the documentation[1] of `numpy.linalg.lstsq()` and found that first the function returns more additional informations, such as *residual sum of squares (or* the difference between the measured and projected point*)*, *rank* and *singular value*s of A. Moreover, it also said that:

> If there are multiple minimizing solutions, the one with the smallest 2-norm $||x||$ is returned.

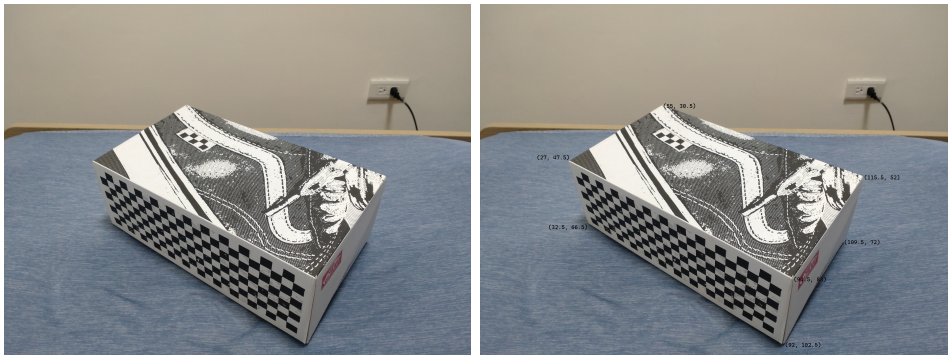
For the purpose of doing so, the matrix needs be examined and sometimes solved by means of **Pseudo Inverse**, while there are faster approach such as **QR decomposition**.

Nevertheless, the returned matrix of `numpy.linalg.lstsq()` tended to be more stable.

[1] *numpy.linalg.lstsq — NumPy v1.23 Manual.* URL: `https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html`.

3 Take a Photo of a Rectangular Object

Figure 1a is a shoe box I chose of size 33 by 18.5 by 12 cm. that is suitable for this task. The 2D coordinates of its vertices is labeled on Figure 1b by hands.



(a) Original photo (b) Marked by hands

Figure 1: The shoe box

Despite the fact that the box isn't cubic, we could still project the box to the unit cube in the first octant since we could consider the project matrix also performs the *scaling* linear transformation. I constructed Table 1 of the corresponding of 3D points between 2D also by hands.

Table 1: Corresponding of 3D Points Between 2D

3D	2D
(0, 0, 1)	(32.5, 66.5)
(0, 1, 0)	(55, 30.5)
(0, 1, 1)	(27, 47.5)
(1, 0, 0)	(109.5, 72)
(1, 0, 1)	(92, 102.5)
(1, 1, 0)	(115.5, 52)
(1, 1, 1)	(94.5, 83)

3.1 Compute the Projection Matrix

By referring the template codes providing, we have the projection matrix computed:

$$P = \begin{pmatrix} 27.32705956 & -12.59046806 & -27.92830833 & 59.3337975 \\ 7.42640673 & -24.90881322 & 7.17919923 & 50.8301141 \\ -0.20857665 & -0.15012128 & -0.15302608 & 1. \end{pmatrix}$$

3.2 Draw the 3D Model

Calling the function provided in template codes, we could obtain the 3D model of the shoe box shown in Figure 2.

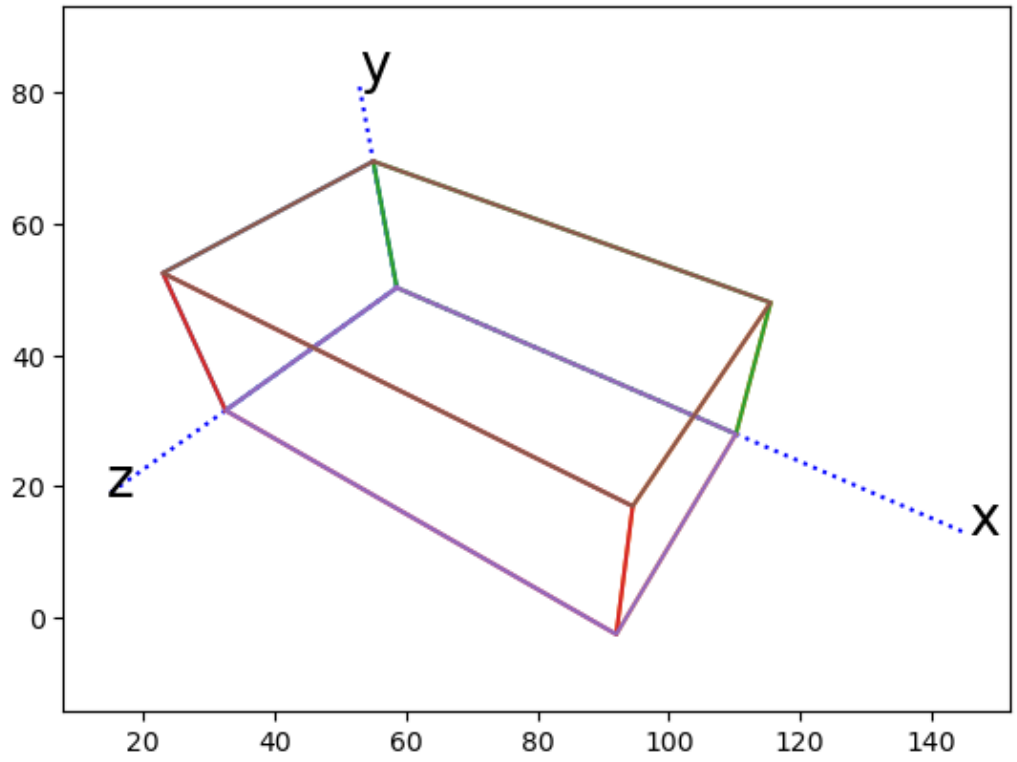


Figure 2: The 3D model of the shoe box

4 Algorithm Related to Single View Geometry

4.1 Derivations

When it comes to camera calibration of an image, there are 3 coordinate systems involved — camera (2D or 3D), image and world.

On the image plane, the points are mapped to 2D camera coordinates by the **camera calibration matrix** C, which is upper-triangular and determined by the scaling α in the image in the two directions and the principal point (u_0, v_0),

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix}$$

.

By *pinhole imaging* and *similar triangles*, we could found the perspective projection from 2D camera to 3D $\begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix} = \lambda \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$, where $\lambda = \frac{f}{Z_c}$, or

$$\begin{pmatrix} x_c \\ y_c \\ f \end{pmatrix} = \begin{pmatrix} I & \mathbf{0} \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix}$$

in *homogeneous system*.

The relation between the camera and world 3D coordinates could be expressed as $\mathbf{X_c} = R\mathbf{X_w} + T$, i.e.,

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R & T \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

, where R is a 3×3 rotation matrix.

As a consequence, we have the transformation between the image and the world coordinate:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = C \begin{pmatrix} I & \mathbf{0} \end{pmatrix} \begin{pmatrix} R & T \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = C(R|T) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

. Thereby we define the 3×4 **projection matrix** $P = C(R|T)$, where $\mathbf{x} = P \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}$. The null-space of P is the optical centre of the camera.

4.2 Processes

By enough correspondences, i.e., at least 6 points, **projection matrix** P could be resolved. Subsequently, we should factorize P back to **camera calibration matrix** C, R and T.

The first 3×3 submatrix of P is the product of an upper triangular matrix C and rotation matrix R. Thus by **QR decomposition**, C and R could be solved. Then for the remaining part, we could find that $T = C^{-1} \begin{pmatrix} p_{14} \\ p_{24} \\ p_{34} \end{pmatrix}$.

Note that C solved by this manner, we would have an additional entry k at $(1, 2)$, indicating the *skew coefficient*.

5 Automatically Select the Corresponding Points

5.1 Procedures

I found this document[2] being of help toward this task online, looked into the computer vision library **OpenCV** and made use of the function `cv2.goodFeaturesToTrack()` to apply *Shi-Tomasi* algorithm[3] to find desired corners in the image.

First I preset the threshold and minimum distance, transformed the image into gray scale and find the 7 most-likely corners, or vertices. Then, I assume that the rightmost point on 2D must be $(0, 0, 1)$ in 3D, says *pa* , the topmost one must be $(0, 1, 0)$, says *pb* and the bottommost and the right most must be $(1, 0, 0)$ and $(1, 0, 1)$, says *pc* and *pd* respectively. For the remaining 3 points, if its x-distance to *pa* is less than the one o the minimum of *pa*, *pc*, then I assume it must be $(0, 0, 0)$ in 3D. Else if its x-distance to *pd* is less than the one to the maximum of *pa*, *pc*, then I assume it must be $(0, 0, 0)$ in 3D. Else if its x-distance to *pa* is less than the one to the minimum of *pa*, *pc*, then I assume it must be $(0, 1, 1)$ in 3D. Otherwise, I assume the point to be $(1, 1, 1)$ in 3D.

[2] *OpenCV: Shi-Tomasi Corner Detector & Good Features to Track*. URL: `https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html`.

[3] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: `10.1109/CVPR.1994.323794`.

5.2 Outcomes

Nevertheless, I tried several photos in real-life including the one for the third question yet their results were all quite awful. As a consequence, I turned to some simple 3D model images generated by some tools and found that my codes worked indeed. Furthermore, I tested a screenshot, shown in Figure 3, taken in the game Minecraft.

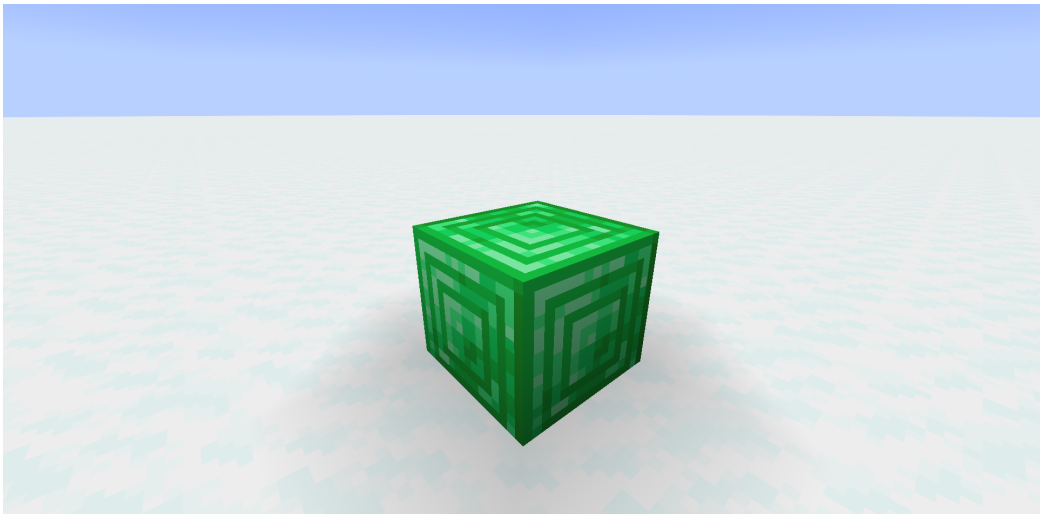


Figure 3: The screenshot of a block in Minecraft

I believe that it may be the inevitable vast amount of noises plus the complex inhered informations for the real-life photos such that my codes couldn't give ideal outcome. It might require some additional effort such as proper preprocessing, Gaussian blurring, but it's supposed to be beyond this course, Linear Algebra.

Although we could see that in Figure 4 the 3D point $(1, 1, 1)$ wasn't marked precisely, yet in my opinion, the model in Figure 5 looked fine enough. And I found that the results might differ, despite the fact that I also tried to fix the selection slice of equations (rows of A, b). Sometimes the result went quite weird. In these cases, we might need rerun cell 28, 29 again.

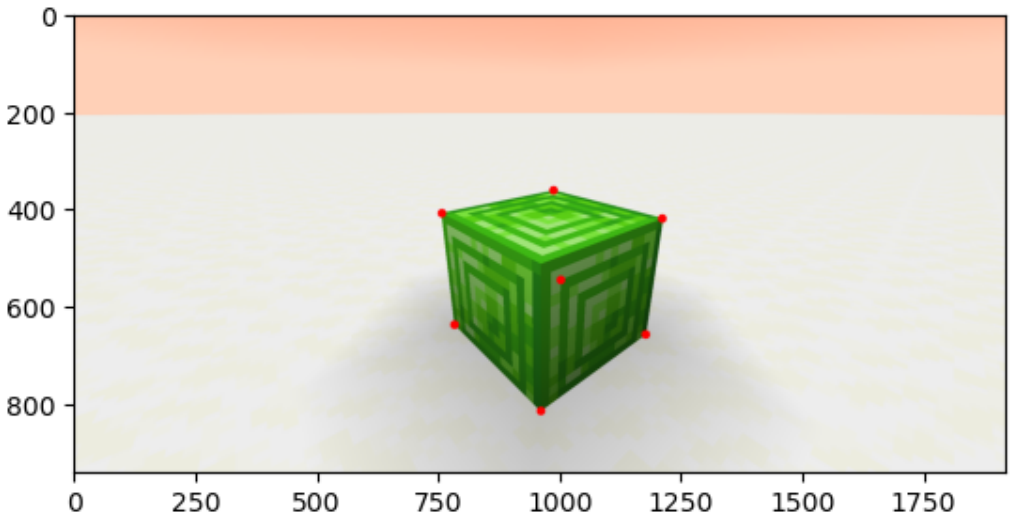


Figure 4: The detected corners, or vertices of the block are dotted

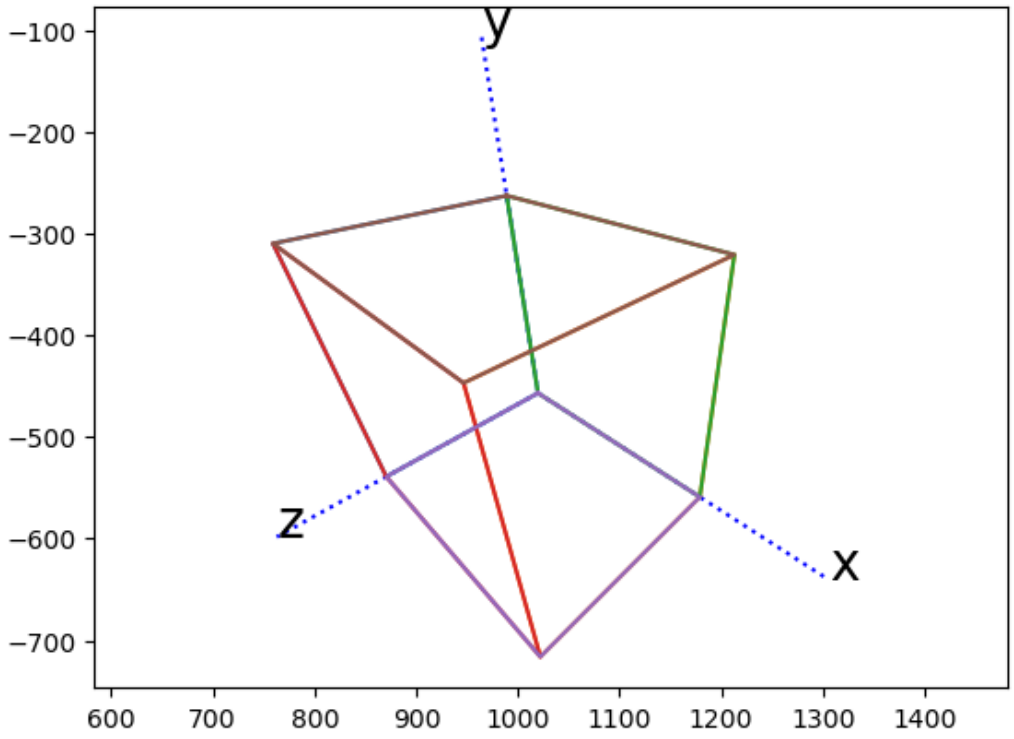


Figure 5: The 3D model of the block

Acknowledgements

I thank to National Center for High-performance Computing *(NCHC)* for providing computational and storage resources.

References

[1] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[2] *numpy.linalg.lstsq — NumPy v1.23 Manual*. URL: https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html.

[3] *OpenCV: Shi-Tomasi Corner Detector & Good Features to Track*. URL: https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html.

[4] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[5] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[6] Andrew Zisserman. *Camera Calibration*. Apr. 1997. URL: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/EPSRC_SSAZ/node5.html#SECTION00050000000000000000.

[7] Andrew Zisserman. "Geometric Framework for Vision I: Single View and Two-View Geometry". In: *Lecture Notes, Robotics Research Group, University of Oxford* (Apr. 1997). URL: http://cg.elte.hu/~hajder/latas/epipolar.pdf.